ДОДАТОК 4

Код клієнт-серверного застосунку

```
const http = require('http');
const express = require('express');
const cors = require('cors');
const io = require('socket.io');
const cookieParser = require('cookie-parser');
const bodyParser = require('body-parser');
const SensorsController = require('./controllers/sensors');
const SocketService = require('./services/socket');
const config = require('./libs/config');
const log = require('./libs/log');
const db = require('./libs/db');

// init app
const app = express();
app.use(cors());
app.use(cookieParser());
app.use(bodyParser.json({ limit: '250mb' }));

// init app http controllers
const router = express.Router();
new SensorsController(router, 'sensors');
app.use(router);

// init app http error handler
app.use((err, req, res) => {
	log.error('Occurs an error:', { path: req.url, stack: err.stack });
	res.status(err.status || 500);
	res.send(err.message || 'Occurs internal server error.');
});

// init app
(async () => {
	// sync database
	await db.sync();

	// init app http server
	const server = http.createServer(app);
	const port = process.env.PORT || config.server.port;
	server.listen(port, () => {
		log.info(`Server listening on: ${port}.`);
	});

	// init app socket server
	const socket = io(server);
	new SocketService(socket);
})();
```

```
// socket service
const SensorsModel = require('./../models/sensors');

// define constants
const MESSAGE_SET_SENSORS_DATA = 'solar/server/SET_SENSORS_DATA';

/**
* Socket service
*/
class SocketService {
	constructor(socket) {
		// handle singleton
		if (SocketService.singleton) {
			return SocketService.singleton;
		}

		// handle connection
		this.socket = socket;
		this.socket.on('connection', (client) => {
			new ClientHandler(client);
		});

		// save singleton
		SocketService.singleton = this;
	}

	/**
	* Method for broadcasting to all user actual sensors data
	*/
	async broadcastSensorsData() {
		const sensorsData = await getSensorsData();
		this.socket.local.emit(MESSAGE_SET_SENSORS_DATA, sensorsData);
	}
}

/**
* Client handler
*/
class ClientHandler {
	constructor(client) {
		this.client = client;
		this.model = new SensorsModel();
		this.handleConnect();
	}

	/**
	* Method for handling new client connection
	*/
	async handleConnect() {
		const sensorsData = await getSensorsData();
		this.client.emit(MESSAGE_SET_SENSORS_DATA, sensorsData);
	}
```

```
}

/**
* Helper for getting sensors data
*/
async function getSensorsData() {
	const model = new SensorsModel();
	const sensors = Array.from(SensorsModel.SENSORS.values());
	const sensorsData = (await Promise.all(sensors.map(async (sensor) => {
		const { data } = await model.getData({ sensorId: sensor.id, limit: 1 });
		if (data && data[0]) {
			return data[0];
		}
		return null;
	}))).filter(Boolean);
	return sensorsData;
}

module.exports = SocketService;


// database connection
const Sequelize = require('sequelize');
const config = require('./config');
const log = require('./log');

// init database connection
const db = new Sequelize({
	host: process.env.DB_HOST || config.db.host,
	port: process.env.DB_PORT || config.db.port,
	database: process.env.DB_NAME || config.db.database,
	username: process.env.DB_USER_NAME || config.db.username,
	password: process.env.DB_USER_PASSWORD || config.db.password,
	dialect: config.db.dialect,
	operatorsAliases: false,
	logging: false,
});

// test database connection
(async () => {
	try {
		await db.authenticate();
		log.info('Database connection has been established successfully.');
	} catch (e) {
		log.error(e.message, { stack: e.stack });
	}
})();

module.exports = db;
```